

d)

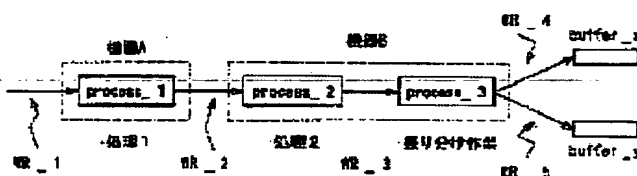
DEVICE AND METHOD FOR PROGRAM GENERATION SUPPORT

Patent number: JP9050371
Publication date: 1997-02-18
Inventor: TERA0 SO; YOSHIKAWA NORIO; MORI KENICHIRO
Applicant: OMRON TATEISI ELECTRONICS CO
Classification:
- international: G06F9/06; G06F17/00; G06F9/06; G06F17/00; (IPC1-7): G06F9/06; G06F17/00; G06F17/60
- european:
Application number: JP19950200954 19950807
Priority number(s): JP19950200954 19950807

Report a data error here

Abstract of JP9050371

PROBLEM TO BE SOLVED: To use the control structure of a model prepared in a virtual environment by a simulator as the upstream design tool of actual system control by structuring a system according to a concept able to be shared with actual systems. SOLUTION: Processes (process 1, process 2, process 3) provided with a data processing content to be executed by equipment in the actual system as processing programs on data in itself and to manage their execution are set from a simulation model from the simulator, and Work routes (WR 1, WR 2, WR 3) are set by designating an order to make Work transit between set processes, and the program of the simulation model is prepared based on the operation setting of set processes and Work routes, and also, an execution program is prepared by linking them.



Data supplied from the esp@cenet database - Worldwide

d)

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-50371

(43) 公開日 平成9年(1997)2月18日

(51) Int.Cl. ⁸	識別記号	序内整理番号	F I	技術表示箇所
G 0 6 F 9/06	5 3 0		G 0 6 F 9/06	5 3 0 T
				5 3 0 H
17/00		7925-5L	15/20	D
17/60			15/21	R

審査請求 未請求 請求項の数18 O L (全 13 頁)

(21) 出願番号 特願平7-200954

(22) 出願日 平成7年(1995)8月7日

(71) 出願人 000002945

オムロン株式会社

京都府京都市右京区花園土堂町10番地

(72) 発明者 寺尾 創

京都府京都市右京区花園土堂町10番地 オムロン株式会社内

(72) 発明者 吉川 典雄

京都府京都市右京区花園土堂町10番地 オムロン株式会社内

(72) 発明者 森 健一郎

京都府京都市右京区花園土堂町10番地 オムロン株式会社内

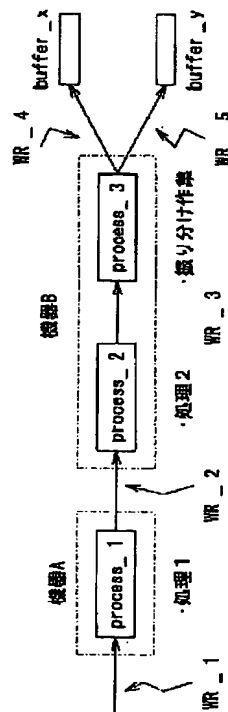
(74) 代理人 弁理士 和田 成則

(54) 【発明の名称】 プログラム作成支援装置およびプログラム作成支援方法

(57) 【要約】

【課題】 実システムとの間で共有できる概念に従ってシステムの構造化を行うことにより、シュミレータでの仮想環境上で作成したモデルの制御構造を実システム制御の上流設計ツールとして利用する。

【解決手段】 シミュレータからのシミュレーションモデルから、実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセス (process_1、process_2、process_3) を設定し、この設定されたプロセス間をWork経路 (WR_1、WR_2、WR_3、WR_4) を設定し、設定されたプロセスおよびWork経路の動作設定に基づき、上記シミュレーションモデルのプログラムを作成するとともに、リンクして実行プログラムを作成する。



【特許請求の範囲】

【請求項1】 シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置であって、
上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をWorkが遷移していく順序を指定するWork経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成手段と、
このコントロールフロー記述プログラム作成手段で記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成する実行コード作成手段とを具備することを特徴とするプログラム作成支援装置。

【請求項2】 上記コントロールフロー記述プログラム作成手段は、
上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定するプロセス設定手段と、
このプロセス設定手段で設定されたプロセス間をWorkが遷移していく順序を指定するWork経路を設定するWork経路設定手段とを有することを特徴とする請求項1記載のプログラム作成支援装置。

【請求項3】 上記実行コード作成手段は、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項1記載のプログラム作成支援装置。

【請求項4】 上記実行コード作成手段は、上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項1記載のプログラム作成支援装置。

【請求項5】 上記実行コード作成手段は、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする請求項1記載のプログラム作成支援装置。

【請求項6】 上記プロセス設定手段は、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする請求項2記載のプログラム作

成支援装置。

【請求項7】 上記プロセス設定手段は、オブジェクト指向言語を使用して上記プロセスをオブジェクトとして形成することを特徴とする請求項2記載のプログラム作成支援装置。

【請求項8】 Work経路設定手段は、上記オブジェクトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする請求項2記載のプログラム作成支援装置。

【請求項9】 上記中間言語は、上記オブジェクト指向言語で記述されることを特徴とする請求項8記載のプログラム作成支援装置。

【請求項10】 シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援方法であって、
上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をWorkが遷移していく順序を指定するWork経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成し、
記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成することを特徴とするプログラム作成支援方法。

【請求項11】 上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定し、このプロセス設定手段で設定されたプロセス間をWorkが遷移していく順序を指定するWork経路を設定することを特徴とする請求項10記載のプログラム作成支援方法。

【請求項12】 上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項10記載のプログラム作成支援方法。

【請求項13】 上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする請求項10記載のプログラム作成支援方法。

【請求項14】 上記コントロールフロー記述プログラ

ムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする請求項10記載のプログラム作成支援方法。

【請求項15】 上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする請求項11記載のプログラム作成支援方法。

【請求項16】 オブジェクト指向言語を使用して上記プロセスをオブジェクトとして形成することを特徴とする請求項11記載のプログラム作成支援方法。

【請求項17】 上記オブジェクトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする請求項11記載のプログラム作成支援方法。

【請求項18】 上記中間言語は、上記オブジェクト指向言語で記述されることを特徴とする請求項17記載のプログラム作成支援方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置およびプログラム作成支援方法に関する。

【0002】

【従来の技術】従来、離散系シミュレータ装置（例えば、米国Deneb Robotics社製のQUEST等）は、最適なシステム構成の検討および稼働予想を事前に行うものである。この従来の離散系シミュレータ装置は、その利用形態として、実在するシステムの解析に限られ、機器ブロックの接続や組み合わせを検討する程度にとどまっていた。

【0003】すなわち、この離散系シミュレータ装置は、システム全体を見通した制御の構造化／設計といった開発上流工程での利用には援用されなかったし、また、シミュレーションモデルが実機制御用プログラムとは別々の概念で構築されているため、互いに互換性がなかった。

【0004】

【発明が解決しようとする課題】しかしながら、このような従来の離散系シミュレータ装置では、実機システムのソフトウェアを開発するに際し、システム全体を見通す必要がある開発上流工程（例えば、構造化／設計の工程）において直接利用できないので、この実機システムを良く理解した人間が時間をかけ、目的とするシステムを分析し設計しなければならず、多大な時間を要するという問題点があった。

【0005】また、シミュレーションモデルが実機システム制御用プログラムとは別々の概念で構築され、互いに互換性がないので、実機システムのプログラムを流用

することができず、ソフトウェアの生産性が悪いという問題点があった。

【0006】特に、近年、実機システムは、ますます大規模化しており、このため所定時間内にシステムを構築するためには、多くの人手をかけなければならないようになってきた。

【0007】そこで、本発明は、上述の問題点に鑑み、実機システムとの間で共有できる概念に従ってシステムの構造化を行うことにより、仮想環境上でのシミュレータにより作成したモデルの制御構造を実機システム制御の上流設計ツールにおいて利用するプログラム作成支援装置およびプログラム作成支援方法を提供することを目的とする。

【0008】

【課題を解決するための手段】前述した目的を達成するため、請求項1記載の発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援装置であって、上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をWorkが遷移していく順序を指定するWork経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成手段と、このコントロールフロー記述プログラム作成手段で記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成する実行コード作成手段とを具備することを特徴とする。

【0009】請求項2記載の発明は、請求項1記載の発明において、上記コントロールフロー記述プログラム作成手段が、上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定するプロセス設定手段と、このプロセス設定手段で設定されたプロセス間をWorkが遷移していく順序を指定するWork経路を設定するWork経路設定手段とを有することを特徴とする。

【0010】請求項3記載の発明は、請求項1記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【0011】請求項4記載の発明は、請求項1記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムに、エディタで記述された上

記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【0012】請求項5記載の発明は、請求項1記載の発明において、上記実行コード作成手段が、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする。

【0013】請求項6記載の発明は、請求項2記載の発明において、上記プロセス設定手段が、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする。

【0014】請求項7記載の発明は、請求項2記載の発明において、上記プロセス設定手段が、オブジェクト指向言語を使用して上記プロセスをオブジェクトとして形成することを特徴とする。

【0015】請求項8記載の発明は、請求項2記載の発明において、Work経路設定手段が、上記オブジェクトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする。

【0016】請求項9記載の発明は、請求項8記載の発明において、上記中間言語が、上記オブジェクト指向言語で記述されることを特徴とする。

【0017】請求項10記載の発明は、シミュレータからシミュレーションモデルを受け、実システムを稼働させる実行コードを作成するプログラム作成支援方法であって、上記シミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスと、このプロセス間をWorkが遷移していく順序を指定するWork経路とからなるコントロールフローを記述したコントロールフロー記述プログラムを作成し、記述されたコントロールフロー記述プログラムに、上記プロセスが処理する処理プログラムを設定して上記実システムを稼働させる実行コードを作成することを特徴とする。

【0018】請求項11記載の発明は、請求項10記載の発明において、上記シミュレータからのシミュレーションモデルから、上記実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するプロセスを設定し、このプロセス設定手段で設定されたプロセス間をWorkが遷移していく順序を指定するWork経路を設定することを特徴とする。

【0019】請求項12記載の発明は、請求項10記載の発明において、上記コントロールフロー記述プログラムに、ライブラリに格納された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼

働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【0020】請求項13記載の発明は、請求項10記載の発明において、上記コントロールフロー記述プログラムに、エディタで記述された上記プロセスが処理する処理プログラムをインクルードして上記実システムを稼働させる実システム稼働プログラムを作成し、これをコンパイルし、さらにリンクして上記実行コードを作成することを特徴とする。

【0021】請求項14記載の発明は、請求項10記載の発明において、上記コントロールフロー記述プログラムをコンパイルし、これに上記プロセスが処理する処理プログラムをリンクして上記実行コードを作成することを特徴とする。

【0022】請求項15記載の発明は、請求項11記載の発明において、上記シミュレータと上記実システム間で共有する論理的工程をプロセスとすることを特徴とする。

【0023】請求項16記載の発明は、請求項11記載の発明において、オブジェクト指向言語を使用して上記プロセスをオブジェクトとして形成することを特徴とする。

【0024】請求項17記載の発明は、請求項11記載の発明において、上記オブジェクトのプロセス機能を実行可能なプログラム言語で記述する際に、中間言語を介してパラメータを入力するパラメータ入力手段を有することを特徴とする。

【0025】請求項18記載の発明は、請求項17記載の発明において、上記中間言語が、上記オブジェクト指向言語で記述されることを特徴とする。

【0026】本発明によれば、シミュレータからのシミュレーションモデルから、プロセスと、Work経路とからなるコントロールフロー記述プログラムを作成する。そして、この作成されたプログラムに、プロセスが処理すべきデータ処理内容を記述した処理プログラムを設定して実システムを稼働させる実行コードを作成する。

【0027】

【発明の実施の形態】以下、本発明に係るプログラム作成支援装置およびプログラム作成支援方法の実施形態を図面に基づいて説明する。

【0028】図1はプログラム作成支援装置の一実施形態の構成を示すブロック図である。この実施形態のプログラム作成支援装置1は、シミュレータ2から得たシミュレーションモデルに基づいて後述するプロセスおよびWork経路とからなるコントロールフローを記述したプログラム（以下、コントロールフロー記述プログラムという）を作成するコントロールフロー記述プログラム作成部11と、後述するようにして実システムを稼働させるテキストでなる実システム稼働プログラムを作成す

る実システム稼働プログラム作成部12と、実システム稼働プログラムをコンパイルしてこのオブジェクトモジュールを作成するコンパイラ13と、実システム稼働プログラムのオブジェクトモジュールをリンクして実システムを稼働させる実コードを作成するリンクカ13とから構成されている。

【0029】ここで、プロセスとは実システム中の機器において実行すべきデータ処理内容を自身内のデータ上に処理プログラムとして有しその実行を管理するものといい、Work経路とはプロセス間をWorkが遷移していく順序を指定する経路をいう。

【0030】実システム稼働プログラム作成部12が、実システム稼働プログラムを作成する内容を詳細に説明する。

【0031】実システム稼働プログラム作成部12は、コントロールフロー記述プログラムに、処理プログラムインクルードライブラリ16に格納されている各プロセスが処理する処理プログラムをインクルードし、上記実システム稼働プログラムを作成するか、または、コントロールフロー記述プログラムに、エディタ15で記述された各プロセスが処理する処理プログラムをインクルードして実システム稼働プログラムを作成する。

【0032】次に、本発明に係るプログラム作成支援装置は、「プロセス」という概念を基本にして制御モデルを表現するので、以下、その内容を説明する。

【0033】説明を簡単にするため、図2に示すようなシステム構成の場合を例にとり説明する。

【0034】このシステムは、機器Aと、機器Bと、buffer_Xと、buffer_Yとで構成されており、機器AにおいてWorkに対するある所定の処理1を行い、次に、機器BにおいてWorkに対するある所定の処理2を行った後、Work_X、Yをそれぞれbuffer_X、Yに振り分ける作業を行っているものとする。

【0035】このシステムに「プロセス」の概念を割り当てると、図3に示すように、「プロセス」というオブジェクト（図中ではprocessと表記されている）間を、Work転送の流れ（以下、Work経路といい、図中ではWork_*と表記されている）で結んだモデルに表現することができる。

【0036】すなわち、図2に示したシステムは、機器Aがprocess_1、機器Bがprocess_2およびprocess_3に表現され、また、process_1に流入するWR_1、process_1から流出してprocess_2に流入するWR_2、process_2から流出してprocess_3に流入するWR_3、process_3から流出してbuffer_Xに流入するWR_4および、process_3から流出してbuffer_Yに流入するWR_5とするモデルで表現することができる。

【0037】なお、このモデルを構成するオブジェクトは、論理的な内容を区切りとして設定されているので、上述した機器Bについてのように、機器の数と一致している必要はない。

【0038】また、Work経路は、矢印で示された順序であるが、常にWorkの実移動を伴うものとは限らない。例えば、機器Bではprocess_2の終了後、Workを同じ位置に固定したままの状態ではprocess_3が起動される。この場合には、Workの実移動はないが、論理上ではWorkがprocess間を遷移するということで（図中のWR_3）定義される。

【0039】ここで、従来より使用されていたベトリネット表現を用いてプロセスの具体例を説明する。

【0040】上記図2中のシステム中における機器Aの動作をベトリネットで表現すると、図4に示すようになる。

【0041】すなわち、『機器Aの稼働状態』および『Workに対する処理の進行状態』がマーキングで表現されることになる。しかし、この表現方法では、Workに対する機器の実行すべき処理内容を記述する部分が存在しない。従って、実システムで利用することができる制御構造を記述するためには、少なくとも処理の内容を記す枠組みが必要となる。

【0042】そこで、図5に示すように、工程の『処理内容』を『処理プログラム』と定義して追加する。

【0043】つまり、『機器Aでの作業』を一般化して『工程』と表現しているのである。なお、図中の『処理プログラム』がプレースP_3で実行される作業内容である。ここで、作業内容としては、例えば、Workを切る、曲げる、穴を開けるなど必要な処理が表現されることになる。

【0044】さらに、図5を、機器Aから『PULL型機器に対する出力』も考慮して一般化すると、図6に示すようになる。

【0045】つまり、①機器Aの作業を『工程』とし、②『Work：処理待ち』プレースを汎用的起動要素としての『処理待ち』とするような一般化を行い、さらに、『PULL型機器』への出力に対応のため、プレース「出力」、「出力要求」およびゲート機構を有するトランジション「T_3」を追加した。

【0046】このようなモデル構成において、特にWork転送機能および「処理プログラム」の実行管理機能の部分をメソッドとしてもつオブジェクトを、クラス「process」のインスタンス・オブジェクトと定義することができる。

【0047】上記の点を考慮に入れて、次に、図3のシステムをオブジェクト指向言語「C++」を用いて表現してみると図7および図8に示した記述になる。

【0048】この例において、オブジェクト「proc

ess]は、図3に示したペトリネットの内容どおりに動作するよう記述されている。

【0049】なお、機器Bの「process Y」は、「PULL型」入力であり、また、「process Z」は、Workの物理的移動の伴わない「PUSH型」とした。

【0050】以下、このプログラムの各部を説明する。

【0051】

【演算子定義と中間言語表記】について

中間言語表記とは、「process 型」オブジェクトへのメッセージ通信を、「process 型」に対して再定義された演算子（図7中の符号71部分参照）を用いて記述するものである。

【0052】このような中間言語表記においては、「process 型」のX、Yおよび「VECT 型」のZに対して以下のような演算が可能となる。

【0053】すなわち、処理を実行する場合においては、 $X+Z$ ：Xが処理を実行した後にWorkを保持し（出力要求待ち状態）、 $X>Y$ ：XからYの順に処理を続けて実行するようになる。また、Workを転送する場合においては、 $X<<Y$ ：XがYにWorkの転送を要求（PULL）し、 $X>>$ ：XGAWorkを強制転送（PUSH）するようになる。

【0054】このような、中間言語表記を用いることにより、図8に示した機器制御に関する記述がより簡素な表現となっていることは、図9（中間言語表記を用いなかった場合の機器制御に関する記述部分）との比較により明らかである。

【0055】

【共通定義】について

ラベル定義では「VECT 型」に相当する定数値を設定している。また、「VECT 型」変数IO_A、IO_Bは、『センサ』などにおいて物理的なWork入力を検出した際、 $0x00 \rightarrow 0x01$ と変化するものと定義する。さらに、このプログラムにおいては、説明を簡単するため、その値が読み出されるまでラッチされているものとする。

【0056】

【機器Aの制御】について

機器Aにおいて、物理的なWorkの入力は、IO_Aをどうして通知され、「process X」は、『Work：処理待』状態となったことを認識する（ $X+IO_A$ ）。すると、「process X」は、『工程：IDLE』状態となり次第、プログラム実行を開始する。また、後続が「PULL型」なので、出力に関する制御は不要となる。

【0057】

【機器Bの制御】について

機器Bにおいて、「process Y」は、『工程：IDLE』状態になると、「process X」にW

orkを出力させる（ $Y<<X$ ）。次に物理的なWork入力は、IO_Bを通して通知され、「process Y」は、『Work：処理待』状態になったことを認識する（ $Y+IO_B$ ）。

【0058】次に、「process Z」は、「process Y」から処理終了を通知され、『Work：処理待』状態となったことを認識する（ $Y+IO_B>Z$ ）。

【0059】このことにより、「process Z」は、物理的な移動を伴わずにWorkを受けとり、「process Y」に連続して処理を開始できる。

【0060】続いて、離散系シミュレータ「QUEST」から「プロセス」により構築されたモデルへのダウンロードについて説明する。

【0061】図10は図2の制御対象を「QUEST」によりダウンロードする際の説明を示すモデル構築図である。

【0062】ここで、制御対象を図2に示した機器Aおよび機器Bとし、かつ、その機器Aをバッファ、機器Bをワーク・セルとするものとする。

【0063】図に示すように、機器Aは、キュー型のバッファであり、機器Bは、1つの入力、2つの出力を有する「PULL型」のワーク・セル（加工機）であるとする。

【0064】そして、ワークセルの動作ロジックは、上述の条件に基づき、機器Aについてはキュー処理（先入れ先出し方式）を行うFIFO、機器Bについて加工に関するSpecify、また、振分け作業に関するものをFixedとして設定する。このSpecifyが、加工の実行を行い、このFixedは、2つの出力とWork種別との振り分けを設定している。

【0065】次に、「中間言語」の仕様を用いた記述を図11に示す。これは、図8と同様なものであるが、「QUEST」の表現と対応したラベル名を使用している。

【0066】また、「VECT 型」変数IO_Aについては、「process Qing」においてバッファ機能を実現するため、処理が任意に実行可能となるよう、物理的なWork入力を検出した際、 $0x00 \rightarrow 0x03$ と変化するものと定義しておく。

【0067】このように、「QUEST」上でのモデル構築で行った各種の設定情報から図11の記述を導くことは、対応関係が明確なので容易である。しかし、図11の記述だけでは、制御内容が不十分であり、「QUEST」においてはメニュー選択で設定した「FIFO」などのWorkに対応する具体的処理が実行できない。

【0068】従って、これらの部分を次のように各「処理カリキュラム」に割り当て、その制御を記述しておく必要がある。

【0069】

すなわち、

```
Void Fifo() {
    F I F O動作を実行するためのハードに依存した制御
}
Void Spfy() {
    W o r k加工動作を実行するためのハードに依存した制御
}
Void Fixdy() {
    W o r k出力振分け動作を実行するためのハードに依存した制御
    ただし、出力タイミング制御のみは、オブジェクト「R o u t」内
    で行なう
}
```

【0070】上述の制御は、採用する機器に依存した制御プログラムとなるので、その内容を状況に応じて記述する必要があるが、「F I L O」などの代表的な動作についてはライブラリとしておけばよい。

【0071】そして、最後に、これらの「中間言語表記」による記述を「C++コンパイラ」で処理するれば、実システムの機器で実行可能なコードが生成される。

【0072】つまり、離散系シミュレータから実システムに、ダウンロードが完了することになる。

【0073】上述した実施形態のプログラム作成支援装置では、実システム稼働プログラム作成部12で、エディタ15または処理プログラムインクルードライブラリ16で作成された各プロセスの処理プログラムを、コントロールフロー記述プログラムにインクルードし、これをコンパイラ13でコンパイルし、さらにリンク14でリンクするような構成になっているが、それ以外として、図12に示すような構成をなしてもよい。

【0074】すなわち、図12に示すプログラム作成支援装置は、シミュレータ2から得たコントロールフロー記述プログラムを作成するコントロールフロー記述プログラム作成部11と、コントロールフロー記述プログラムをコンパイルしてこのオブジェクトモジュールを作成するコンパイラ13と、処理プログラムリンクライブラリ17に格納された各プロセスが処理する処理プログラムを、コンパイラ13において作成されたオブジェクトモジュールにリンクして実システムを稼働させる実行コードを作成するリンク14とで構成してもよい。

【0075】

【発明の効果】上述の本発明によれば、シミュレータからのシミュレーションモデルから、プロセスおよびW o r k経路が設定されるため、各種離散系シミュレータ間での「制御構造」の共有化および実システムへのダウンロードが可能となり、離散系シミュレータの「制御構造の上流設計ツール」として活用および実システムのプログラムの自動生成が推進できる。

【0076】このため、上流工程のシステム分析・設計を容易にし、また、プログラムの生産性を向上させることができる。

【図面の簡単な説明】

【図1】本実施例のプログラム作成支援装置の一実施形態の構成を示すブロック図。

【図2】実システムの一例を構成を示すブロック図。

【図3】図2に示した構成を「プロセス」を適用したモデルに示したブロック図。

【図4】図2中の機器Aをベトリネット表現で示した図。

【図5】図4に工程処理を含ませた図。

【図6】図2中の機器Aを工程処理および「P U L L型機器への出力」を考慮して一般化した図。

【図7】図3のシステムをオブジェクト指向言語「C++」で記述した図。

【図8】図3のシステムをオブジェクト指向言語「C++」で記述した図。

【図9】中間言語表記を含まない場合の機器制御を記述した図。

【図10】図2の制御対象を「Q U E S T」によりダウンロードする際の説明を示すモデル構築図。

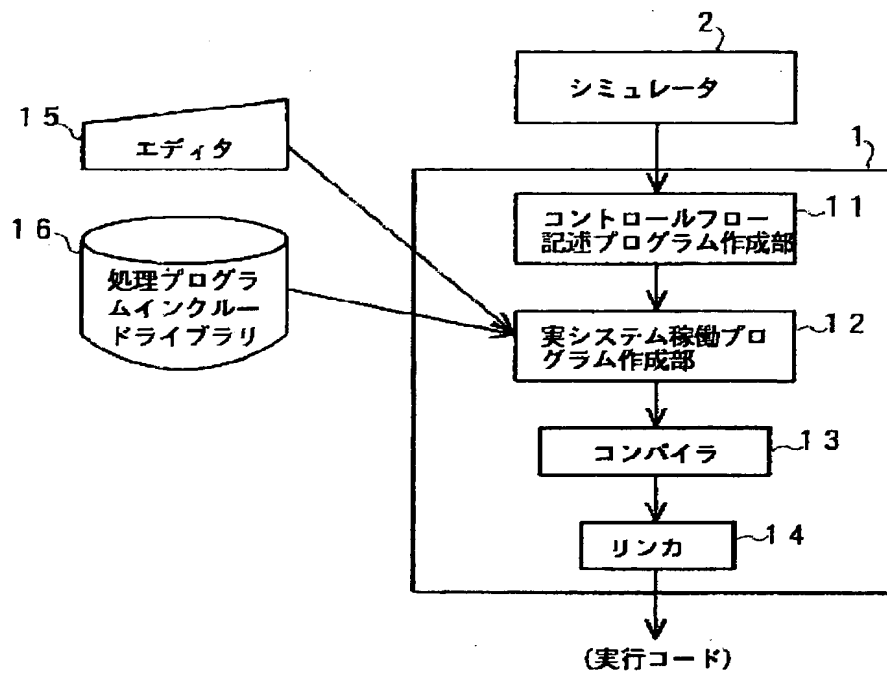
【図11】システムを記述したプログラムのうちの「中間言語表記」を示す図。

【図12】他の実施例のプログラム作成支援装置の一実施形態の構成を示すブロック図。

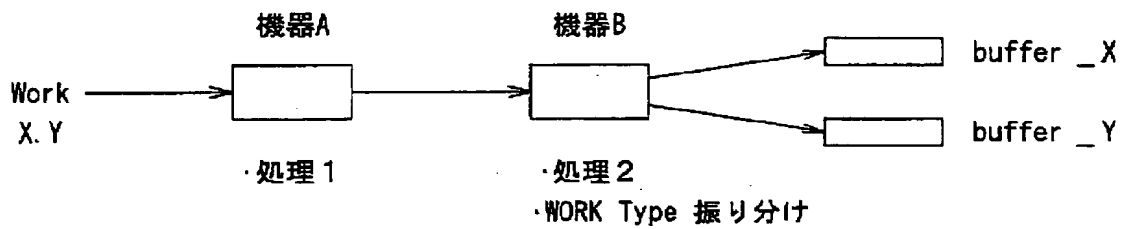
【符号の説明】

- | | |
|----|--------------------|
| 1 | プログラム作成支援装置 |
| 2 | シミュレータ |
| 11 | コントロールフロー記述プログラム作成 |
| 12 | 実システム稼働プログラム作成部 |
| 13 | コンパイラ |
| 14 | リンク |
| 15 | エディタ |
| 16 | 処理プログラムインクルードライブラリ |
| 17 | 処理プログラムリンクライブラリ |

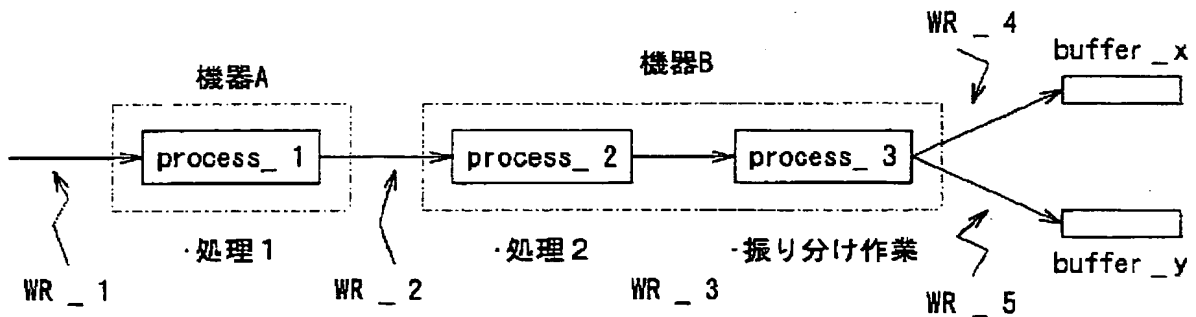
【図1】



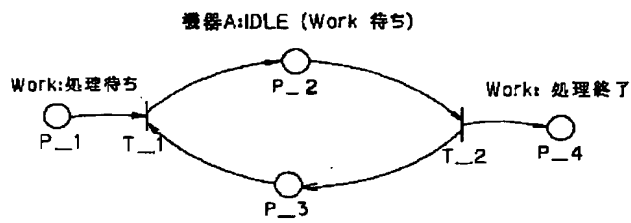
【図2】



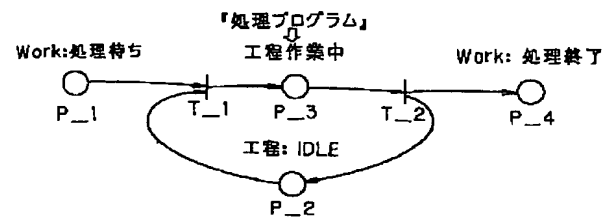
【図3】



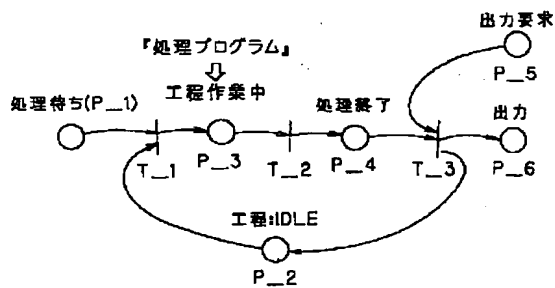
【図4】



【図5】



【図6】



【図8】

【機器制御プログラム】

```

VECT IO_A, IO_B                                // 物理的IO信号(I/O)の入力用

void main_A()                                    // 機器A用main 関数
{
    process X(Code_X);                          // クラス宣言
    while(1) X + IO_A;                          // Xを実行
}
void main_B()                                    // 機器B用main 関数
{
    process Y(Code_Y);                          // クラス宣言
    process Z(Code_Z);                          // クラス宣言

    while(1) {
        Y << X;                                // Xから Work を PULL
        Y + IO_B + Z;                          // Y, Z の順に実行
        Z >> ;                                // Z PUSH 出力
    }
}

```

【図7】

//【共通定義】

```

#define VECT int
#define P_1 0x01 // 『WORK入力』
#define P_2 0x02 // 『工程IDLE』
#define P_4 0x08 // 『処理終了』
#define P_5 0x10 // 『出力要求』

```

//【クラス定義】

```

class process {
    int (*Func) // 「処理プログラム」を記述する関数へのポインタ
    VECT MARKing : // 「process」の状態変数

public:
    process(int (*pFunc)()) Func = pFunc : // object 構築関数
    ~process() : // object 消滅関数
    operator + (VECT) : // 『 + 』演算の定義
    operator * (VECT) : // 『 * 』演算の定義
    operator >> (VECT) : // 『 >> 』演算の定義
    operator << (VECT) : // 『 << 』演算の定義
    operator | (VECT) : // 『 | 』演算の定義
};

process::operator+(VECT inMK) { // 【『 * 』演算の定義】
    MARKing |= inMK : // Token の追加設定
    if (MARKing & 0x03 == 0x03) { // Token : (P_1 & P_2) ?
        MARKing = (MARKing & 0xF0) + 0x04 : // Token - P_3
        (*Func)() : // Func の参照する関数を実行
        MARKing = (MARKing & 0xF0) + 0x08 : // Token -> P_4
    }
    if (MARKing & 0x18 == 0x18) { // Token : (P_4 & P_5) ?
        Out Work() : // 物理的 WORK 出力(外部関数)
        MARKing = (MARKing & 0xE0) + 0x02 : // Token -> P_2
    }
}

VECT process::operator*(VECT inMK) { // 【『 * 』演算の定義】
    inMK &= MARKing : // 入力変数 AND 内部状態
    return(inMK) : //
}

process& process::operator >> () { // 【『 >> 』演算の定義】
    this+P_5 : // PUSH 出力
}

process& process::operator << (process Y) { // 【『 << 』演算の定義】
    if(this*P_2) Y+P_5 : // PULL 出力
}

process& process::operator | (process Y) { // 【『 | 』演算の定義】
    if(this*P_4) {
        this+P_5 : // this: Token -> P_2
        Y+P_1 : // Y: Token -> P_1
    }
}

```

71

【図9】

【機器制御プログラム】

```

void main_A()                // 機器A用 main 関数
{
    process    Y(Code_A);    // クラス宣言

    while(1) {
        if (IO_A) X+P_1;    // Work の物理的入力検知      X: 処理開始
    }
}

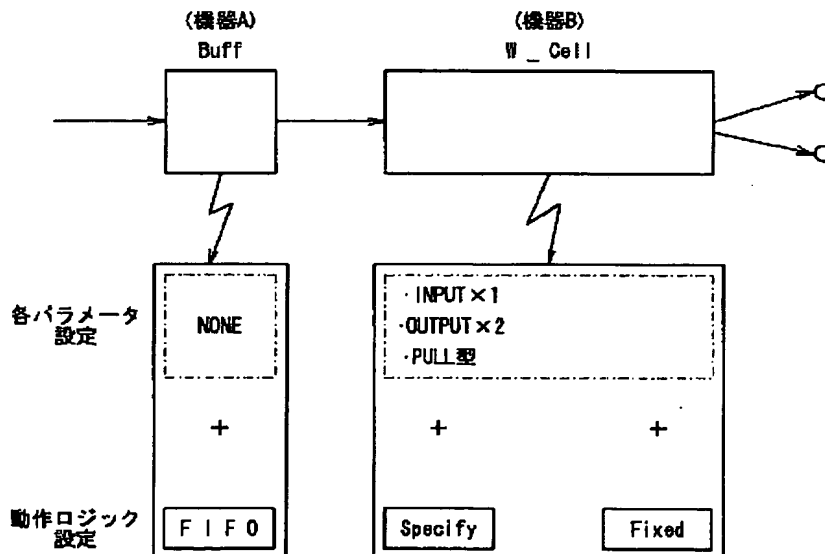
void main_B ()               // 機器B用 main 関数
{
    process    Y(Code_A);    // クラス宣言
    process    Z(Code_B);    // クラス宣言
    VECT       Link = 0;     // 「処理終了」通知用

    while(1) {
        if (Y*P_2) X+P_5;    // Y: IDLE ならば X: PULL 入力
        if (IO_B) {          // Work の物理的入力検知
            Y+P_1;            // Y: 処理開始
            if (Y*P_4) {      // Y: 処理終了 ?
                LINK = 0x01;  // フラグON (Token -) P 1)
                Y+P_5;        // Y: PUSH出力
            }
        }

        Z+Link;               // Z: 処理開始 (Token -) P 1)
        Link = 0x00;          // フラグOFF
        Z+P_5;                // Z: PUSH出力
    }
}

```

【図10】



【図11】

//「中間言語による記序述」

int B_in, W_in;

// Buf, Wcell の入力検知IO

// Buf の制御

void Buf main()

{

process Qing(Fifo);

// クラス宣言

while(1) Qing + IQ_A;

// Qing を実行

}

// W_Cell の制御

void Wcell main()

{

process proc(Spfy);

// クラス宣言

process Rout(Fixd);

// クラス宣言

while(1) {

Proc ({ Qing ;

// Buf から Work をPULL

Proc + IQ_B | Rout;

// Proc, Rout の順に実行

Rout });

// Rout PUSH 出力

}

}

【図12】

